Comparison of Parallel Programming Models on Intel MIC Computer Cluster

CHENGGANG LAI[1], ZHIJUN HAO[2], MIAOQING HUANG[1], XUAN SHI[1] AND HAIHANG YOU[3]

[1]University of Arkansas, [2]Fudan University, [3]Chinese Academy of Sciences

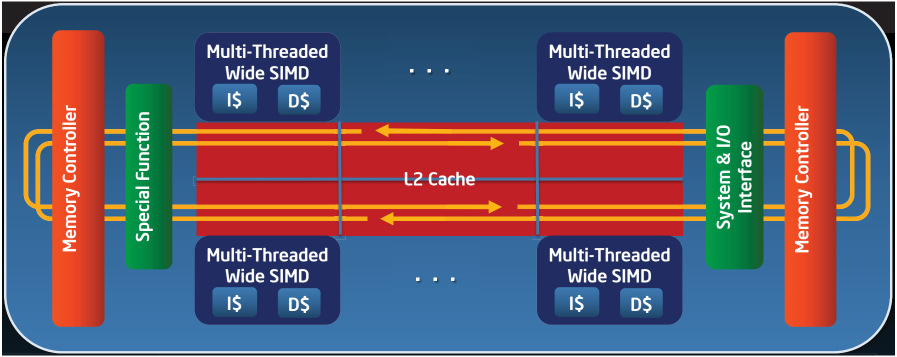
ASHES WORKSHOP, Phoenix May 19, 2014

Outline

Introduction

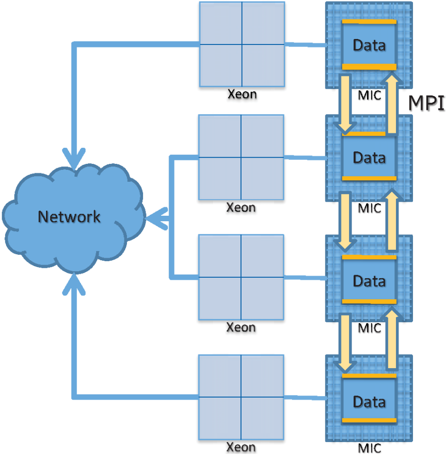
- Accelerators/coprocessors provide a promising solution for achieving both high performance and energy efficiency
  - Intel MIC accelerated clusters: Tianhe-2, Stampede, Beacon
  - GPU accelerated clusters: Titan, Tianhe, Blue Waters
- Multiple parallel programming models on Intel MIC accelerated clusters
  - Native mode
  - Offload mode
  - Hybrid mode
- Use two benchmarks with different communication patterns to test the performance and the scalability of a single MIC processor and an MIC cluster

MIC architecture (Knights Corner)

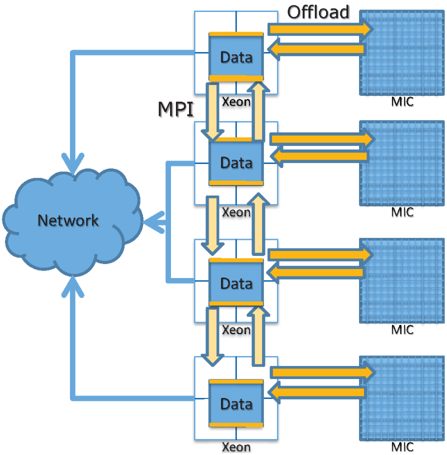


- Contain up to 61 low-weight processing cores
  - Each core can run 4 threads in parallel
- High-speed bi-directional, 1024-bit-wide ring bus
  - 512 bits in each direction

MIC programming models



Native mode

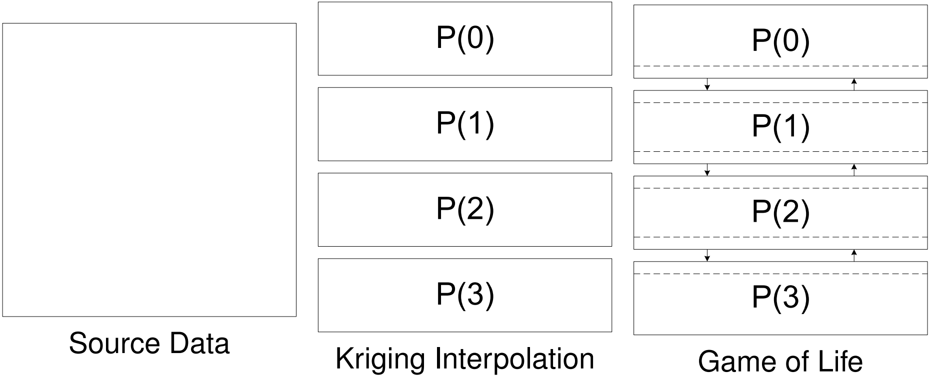


Offload mode

- MPI directly on MIC cores
- MPI on CPUs
  - Offload computation to MIC using OpenMP

Outline

Application communication patterns

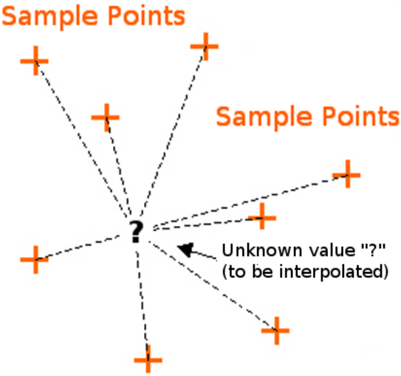


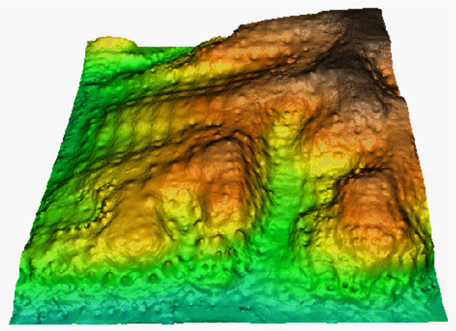
- Kriging interpolation
  - Embarrassingly parallel
- Game of Life
  - Intense communication

Kriging interpolation

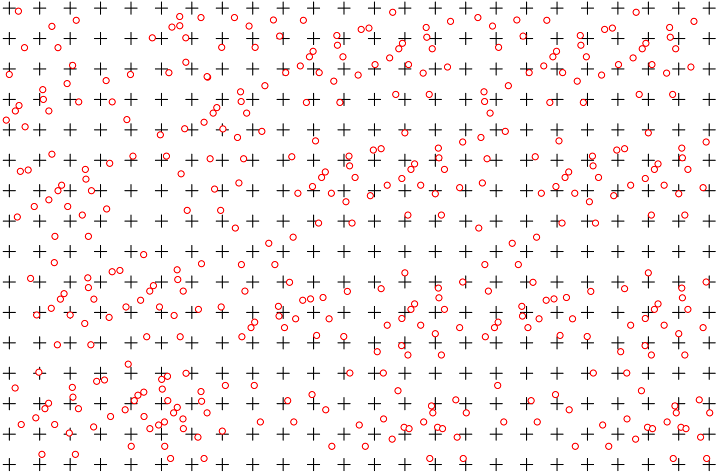
- The value at an unknown point should be the average of the known values of its neighbors

$$\hat{Z}(x, y) = \sum_{i=1}^{k} w_i Z_i$$





Kriging interpolation



- ○: points with known values
- +: points with unknown values to be interpolated

Kriging interpolation benchmark

- Problem size: 171 MB
  - 29 MB: 2,191 sample points
  - 37 MB: 4,596 sample points
  - 48 MB: 6,941 sample points
  - 57 MB: 9,817 sample points
- Output: 4 grids of $1{,}440 \times 720$
  - Use 10 closest sample points to estimate one point in the grid
  - 4 grids are computed in sequence
  - For each grid, the computation is partitioned along the column

Game of Life

- The universe of the GOL is a two-dimensional grid of cells
  - one of two possible states, alive (‘1’) or dead (‘0’)
- Every cell interacts with its eight neighbors to decide its fate in the next iteration of simulation
- The status of each cell is updated for 100 iterations
  - The statuses of all cells are updated simultaneously in each iteration

0	0	1	0	1
0	1	0	1	0
0	0	1	1	0
0	1	1	1	0
1	0	0	0	0

$\longrightarrow$

0	0	1	1	0
0	1	0	0	1
0	0	0	0	1
0	1	0	1	0
0	1	1	0	0

Game of Life

- Rules:
  - Any live cell with fewer than two live neighbors dies, as if caused by **under-population**
  - Any live cell with two or three live neighbors lives on to the next generation
  - Any live cell with more than three live neighbors dies, as if by overcrowding
  - Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction

0	0	1
0	1	0
0	0	0

→

0	0	1
0	0	0
0	0	0

Game of Life

- Rules:
  - Any live cell with fewer than two live neighbors dies, as if caused by under-population
  - Any live cell with two or three live neighbors **lives on** to the next generation
  - Any live cell with more than three live neighbors dies, as if by overcrowding
  - Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction

1	0	1
0	1	0
0	0	0

$\longrightarrow$

1	0	1
0	1	0
0	0	0

Game of Life

- Rules:
  - Any live cell with fewer than two live neighbors dies, as if caused by under-population
  - Any live cell with two or three live neighbors lives on to the next generation
  - Any live cell with more than three live neighbors dies, as if by **overcrowding**
  - Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction

1	0	1
0	1	0
1	1	0

$\longrightarrow$

1	0	1
0	0	0
1	1	0

Game of Life

- Rules:
  - Any live cell with fewer than two live neighbors dies, as if caused by under-population
  - Any live cell with two or three live neighbors lives on to the next generation
  - Any live cell with more than three live neighbors dies, as if by overcrowding
  - Any dead cell with exactly three live neighbors becomes a live cell, as if by **reproduction**

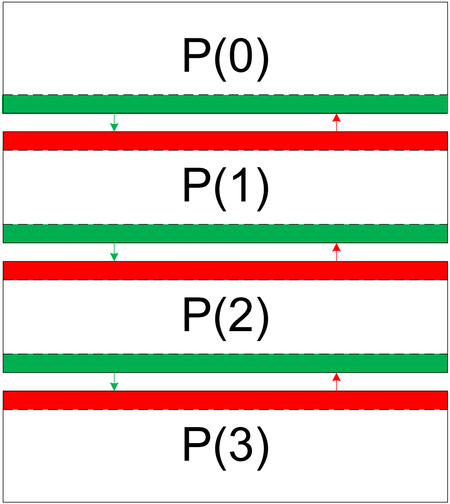
1	0	1
0	0	0
1	0	0

→

1	0	1
0	1	0
1	0	0

Game of Life: communication patterns

- The boundary rows need to be sent to neighbor processing nodes between iterations



Computer platform

- Beacon system
  - A Cray CS300-AC cluster
  - 48 compute nodes and 6 I/O nodes
- Compute node
  - 2 Intel Xeon E5-2670 8-core CPUs
  - 4 Intel Xeon Phi 5110P coprocessors
  - 256 GB RAM
  - 960 GB SSD storage
- Intel Xeon Phi 5110P coprocessor
  - 60 MIC cores at 1.053 GHz
  - 8 GB GDDR5 on-board memory

Outline

Outline

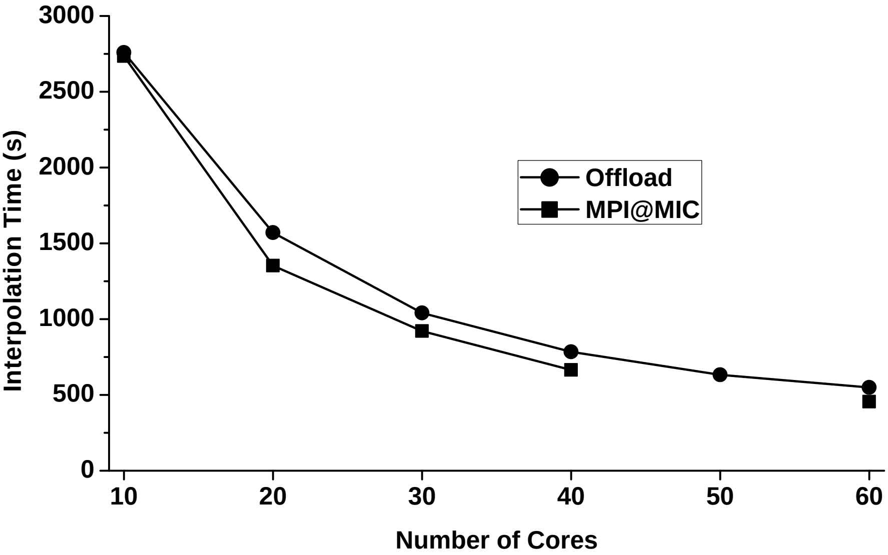
Performance of Kriging interpolation on a single MIC processor (unit: second)

	Number of MIC cores					
Programming model: MPI@MIC						
	10	20	30	40	50	60
Read	0.65	0.60	0.66	0.72	NA*	0.79
Interpolation	2734.45	1353.48	921.76	664.74		455.34
Write	9.44	9.21	11.04	8.04		7.95
Total	2744.54	1363.30	933.46	673.50		464.09
Programming model: Offload						
	10	20	30	40	50	60
Read	0.04	0.05	0.04	0.04	0.04	0.04
Interpolation	2758.22	1570.75	1040.44	784.30	632.65	548.15
Write	1.77	1.99	1.65	1.44	1.45	1.57
Total	2760.03	1572.78	1042.12	785.78	634.14	549.75

*The work could not be distributed into 50 cores evenly.

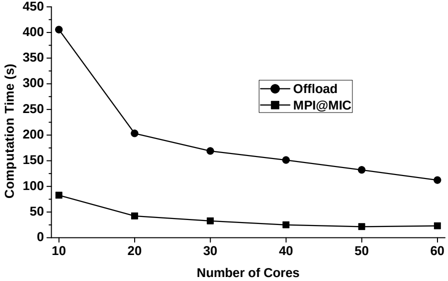
- MPI@MIC
  - The computation of 720 columns is distributed evenly among MPI processes (ranks)
- Offload
  - Use OpenMP to parallelize the `for` loops

Performance of Kriging Interpolation on a single MIC processor

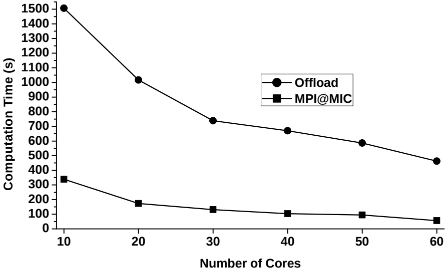


Performance of Game of Life on a single MIC processor (unit: second)

Problem Size	Number of MIC cores					
Programming model: MPI@MIC						
	10	20	30	40	50	60
8192×8192	82.85	42.27	32.56	24.91	21.37	23.15
16384×16384	338.57	173.57	131.10	103.30	94.41	56.31
Programming model: Offload						
	10	20	30	40	50	60
8192×8192	405.35	203.23	168.78	151.34	131.94	112.19
16384×16384	1506.47	1017.12	738.46	670.12	586.65	462.87



8,192×8,192

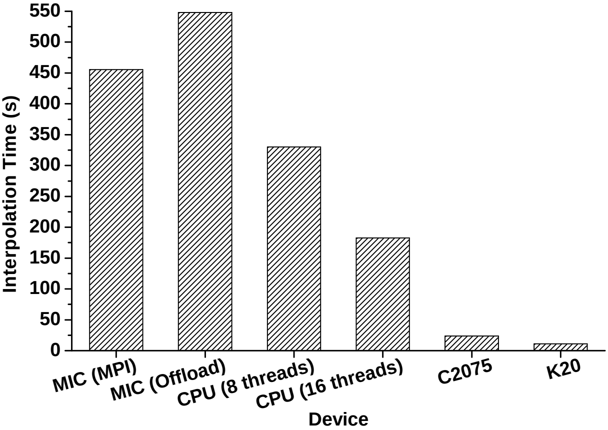


16,384×16,384

Outline

Performance of Kriging interpolation on single devices

	MIC (60 cores)		CPU (Xeon E5-2670)		Nvidia GPU	
	MPI	Offload	8 threads	16 threads	C2075	K20
Read	0.79	0.04	0.01	0.01	0.01	0.01
Interpolation	455.34	548.15	330.11	182.60	23.87	10.90
Write	7.95	1.57	9.85	10.27	1.68	1.68
Total	464.09	549.75	339.96	192.86	25.55	11.77



The performances of MIC and CPU are in the same order of magnitude

Performance of Game of Life on single devices (Unit: second)

	MIC (60 cores)		CPU (Xeon E5-2670)		Nvidia GPU	
	MPI	Offload	8 threads	16 threads	C2075	K20
8192×8192	23.15	112.19	12.03	8.13	15.36	3.25
16384×16384	56.31	462.87	48.22	32.65	58.44	12.58
32768×32768	NA	NA	217.33	114.98	274.03	46.99

- The performance of MPI@MIC: same order of magnitude as CPU and C2075 GPU
- Offload on MIC: one order of magnitude worse
- K20 GPU: one order of magnitude better

Outline

Three parallel programming models

- MPI@MIC
  - MPI-based parallel implementation on Beacon. The Intel Xeon Phi 5110P is used for data processing. In this implementation, each MIC core will directly host one single-thread MPI process. Therefore, if m Xeon Phi coprocessors are used, $m \times 60$ MPI processes are created in the parallel implementation
- MPI@MIC+OpenMP
  - Each MIC core on Intel Xeon Phi 5110P can support up to 4 threads. In this implementation, 4 threads are created in each MPI process running on a MIC core
- MPI@CPU+offload
  - In this implementation, the MPI processes are running on the CPU. The data processing is offloaded to MIC through OpenMP

Outline

1 Introduction

2 Experiment setup

3 Results on single device
- Scalability on a single MIC processor
- Performance comparison of single devices

4 **Results on multiple devices**
- **Comparison among three programming models**
- Experiments on the MPI@MIC+OpenMP programming models
- Experiments on the MPI@CPU+offload programming models
- Experiments on the distribution of MPI processes
- Hybrid MPI vs native MPI

5 Conclusions

Performance of Kriging interpolation under various programming models (unit: second)

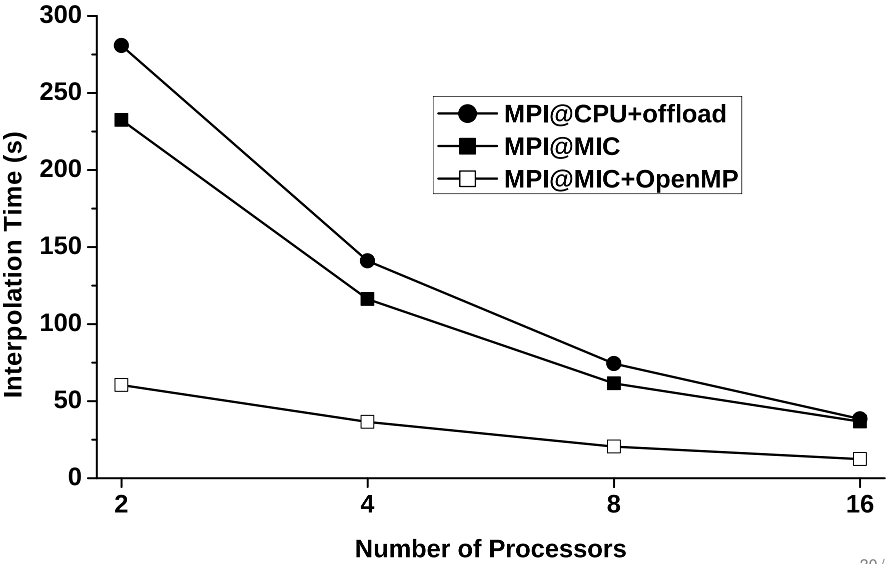
Number of Processors	MPI@MIC				MPI@MIC+OpenMP(4 threads)			
	Read	Interpolation	Write	**Total**	Read	Interpolation	Write	**Total**
2	1.24	232.43	12.24	245.90	0.57	60.43	8.82	69.82
4	1.27	116.34	16.44	134.05	0.51	36.54	122.53	159.59
8	1.23	61.48*	54.43	117.14	0.50	20.43*	240.33	261.26
16	1.31	36.74*	300.23	338.28	0.52	12.33*	210.45	223.30

Number of Processors	MPI@CPU+offload			
	Read	Interpolation	Write	**Total**
2	0.18	280.83	1.60	282.61
4	0.04	141.03	1.27	142.33
8	0.04	74.30	1.19	75.53
16	0.04	38.54	5.94	44.51

*Only 360 or 720 MIC cores are used in the computation with 8 or 16 processors, respectively.

- MPI@MIC+OpenMP: ∼3 times faster than MPI@MIC

Performance of Kriging interpolation under various programming models



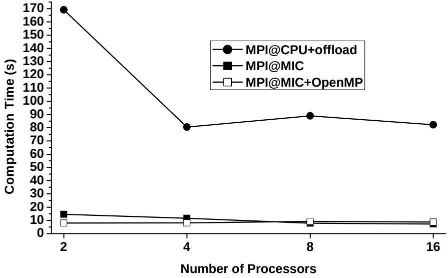
Performance of Game of Life under various programming models (unit: second)

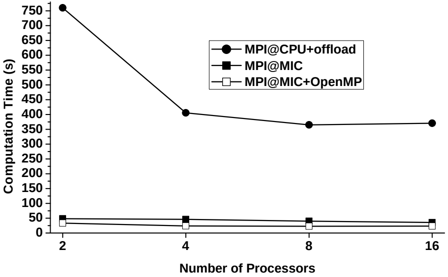
Number of Processors	8,192×8,192			16,384×16,384		
	MPI @MIC	MPI@MIC+ OpenMP(4 threads)	MPI@CPU+ offload	MPI @MIC	MPI@MIC+ OpenMP(4 threads)	MPI@CPU+ offload
2	14.56	7.99	169.12	48.39	33.11	760.20
4	11.63	8.04	80.50	46.31	24.06	405.66
8	7.84	9.28	89.03	39.78	22.98	365.23
16	7.18	8.74	82.51	35.30	23.60	370.65

Number of Processors	32,768×32,768		
	MPI @MIC	MPI@MIC+ OpenMP(4 threads)	MPI@CPU+ offload
2	194.15	149.43	2926.34
4	169.54	104.14	1512.72
8	157.73	106.24	1502.51
16	128.40	110.99	1517.89

- All three programming models lose strong scalability
- It is critical to keep a balance for communication intensive applications

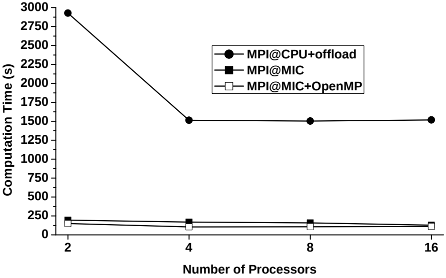
Performance of Game of Life under various programming models





8,192×8,192

16,384×16,384



Outline

Performance of Game of Life using MPI@MIC+OpenMP programming model (Unit: second)

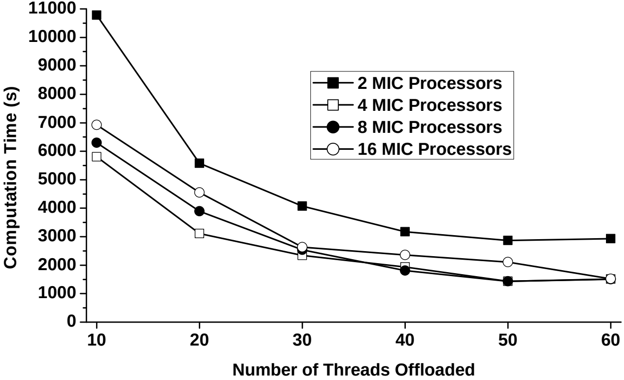
Number of Processors	8,192×8,192		16,384×16,384		32,768×32,768	
	4 threads	8 threads	4 threads	8 threads	4 threads	8 threads
2	7.99	10.94	33.11	32.92	149.43	110.37
4	8.04	9.03	24.06	27.94	104.14	109.79
8	9.28	8.39	22.98	25.69	106.24	100.79
16	8.74	10.77	23.60	27.11	110.99	110.67

- No significant performance improvement for adding more threads on each core

Outline

Performance of Game of Life (32,768×32,768) using MPI@CPU+offload programming model (unit: second)

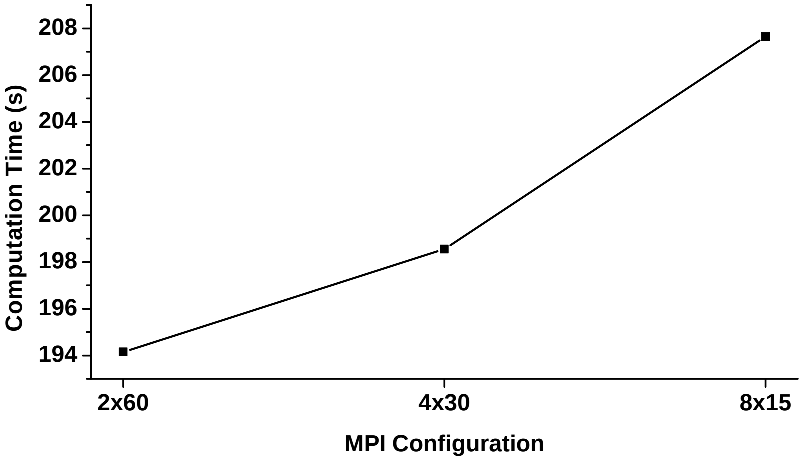
Number of Processors	# of OpenMP threads offloaded to each MIC processor					
	10	20	30	40	50	60
2	10779.47	5578.45	4077.90	3173.22	2870.26	2926.34
4	5807.45	3113.00	2345.75	1935.45	1431.62	1512.72
8	6298.11	3891.83	2540.66	1806.12	1434.91	1502.51
16	6923.38	4549.69	2630.39	2354.70	2104.73	1517.89



More cores do not necessarily bring better performance

Outline

Performance of Game of Life (32,768×32,768) under different MPI configurations (MPI@MIC)



- Inter-card communication takes longer time than intra-card communication

Outline

Hybrid MPI is better than native MPI

- Hybrid MPI
  - MPI processes run on both MIC cores and CPU cores
- Kriging interpolation (57 MB data set) on Beacon
  - 16 MPI processes on one Xeon E5-2670 CPU: 46.02 seconds
  - 16 MPI processes on one Xeon E5-2670 CPU + 14 MPI processes on one MIC card: 24.75 seconds
- Game of Life (16,384 $\times$ 16,384) on a separate workstation
  - 120 MPI processes on two MIC cards: 30 seconds
  - 120 MPI processes on two MIC cards + 12 MPI processes on one Xeon E5-2620 CPU: 27.42 seconds

Outline

Conclusions

- Native mode typically outperforms offload mode
- Further improve the performance by running multiple threads on each MIC core
- Schedule MPI processes to as few MIC processors as possible to reduce the cross-processor communication overhead
- Hybrid mode can outperform native mode